

Norm Verification and Analysis of Electronic Institutions

Wamberto W. Vasconcelos

Department of Computing Science, University of Aberdeen
Aberdeen AB24 3UE, United Kingdom
wvasconcelos@acm.org

Abstract. Electronic institutions are a formalism to define and analyse protocols among agents with a view to achieving global and individual goals. In this paper we propose a definition of norms for electronic institutions and investigate how these norms can be employed for verification and analysis. We offer automatic means to perform the extraction of sub-parts of an electronic institution in which norms hold true or can safely be avoided. These sub-parts can be used to synthesise norm-aware agents that will pursue or avoid commitments to norms.

1 Introduction

An important aspect in the design of heterogeneous multiagent systems (MAS, henceforth) concerns the *norms* that should constrain and influence the behaviour of its individual components [1–3]. Electronic institutions have been proposed as a formalism to define and analyse protocols among agents with a view to achieving global and individual goals [4, 5]. In this paper we propose a definition for norms and a means of using this definition to verify properties of electronic institutions. We also describe means to help designers analyse an electronic institution with a view to extracting alternative and restricted versions of it in which norms are guaranteed to be fulfilled or versions in which norms will never be adopted. We observe that restricted versions of an electronic institution can be used to synthesise agents that will either pursue norms or avoid commitments to norms.

Electronic institutions define *virtual environments* in which agents interact. Designers specify their electronic institutions which may become arbitrarily complex. Tools and mechanisms ought to ensure that certain properties of electronic institutions hold before they can be enacted (*i.e.* agents interact following the specified order and kind of messages of an electronic institution). Some such properties are well-formedness and reachability of all parts of the specification by agents (*i.e.*, absence of “dead parts” that are never used) [6].

Norms, as defined in this work, provide means to check for additional properties of electronic institutions. Our norms are of the kind: if agent x says M_x and agent y says M_y then agent z is *obliged* to say M_z . Given an electronic institution and a set of norms, we want to check if the agents taking part in an enactment of it will indeed abide by the norms prescribed and whether the norms will have any effect on them. We observe that the machinery required for the verification of such properties can also be used to help designers analyse their specification with a view to *extracting* sub-parts of it in which norms are guaranteed to hold (or, alternatively, sub-parts in which agents will not commit to the norms).

Ours is a formal declarative approach. Declarative formal specifications have many advantages [7, 8] over procedural notations. We capitalise on the ability

to use the very same specification to check for properties as well as to obtain execution models of future systems to be devised using the specification [6, 9]. We employ logic programming (in particular, Prolog) [10] to describe all our concepts and proposed functionalities. Although we could have employed “cleaner” formalisms to represent our concepts and solutions, Prolog is a good compromise between a detailed implementation and abstract mathematical formulations.

In Section 2 we introduce a lightweight definition of electronic institutions and a declarative representation for them. In Section 3 we introduce a definition of norms and explain their incorporation to electronic institutions; we also show how norms can be used to check if the agents of an electronic institution will ever commit to a norm and, if they do, whether a norm will eventually be fulfilled. In Section 4 we show how we can analyse electronic institutions with respect to norms in order to extract sub-parts in which norms are fulfilled or never committed to by any agents. We present our conclusions in Section 5, compare our research with related work and give directions for future research.

2 Lightweight Electronic Institutions

Electronic institutions (e-institutions, for short) can be viewed as a variation of non-deterministic finite state machines [11]. We present e-institutions here in a “lightweight” version: those features not essential to our investigation have been omitted. We refer readers to [4, 5] for a complete description of e-institutions.

Our lightweight e-institutions are defined as sets of *scenes* related by *transitions*. We shall assume the existence of a communication language CL among the agents of an e-institution as well as a shared ontology which allow them to interact and understand each other. We first define a scene:

Definition 1. *A scene is a tuple $\mathbf{S} = \langle R, W, w_0, W_f, WA, WE, \Theta, \lambda \rangle$ where*

- $R = \{r_1, \dots, r_n\}$ is a finite, non-empty set of roles;
- $W = \{w_0, \dots, w_m\}$ is a finite, non-empty set of states;
- $w_0 \in W$ is the initial state;
- $W_f \subseteq W$ is the non-empty set of final states;
- WA is a set of sets $WA = \{WA_r \subseteq W, r \in R\}$ where each $WA_r, r \in R$, is the set of access states for role r ;
- WE is a set of sets $WE = \{WE_r \subseteq W, r \in R\}$ where each $WE_r, r \in R$, is the set of exit states for role r ;
- $\Theta \subseteq W \times W$ is a set of directed edges;
- $\lambda : \Theta \mapsto CL$ is a labelling function associating edges to messages in the agreed language CL .

A scene is a protocol specified as a finite state machine where the states represent the different stages of the conversation and the directed edges connecting the states are labelled with messages of the communication language. A scene has a single initial state (non-reachable from any other state) and a set of final states representing the different possible endings of the conversation. There should be no edges connecting a final state to any other state. Because we aim at modelling multi-agent conversations whose set of participants may dynamically vary, scenes allow agents to join or leave at particular states during an ongoing conversation,

depending on their role¹. For this purpose, we differentiate for each role the sets of access and exit states.

To illustrate this definition, in Figure 1 we provide a simple example of a

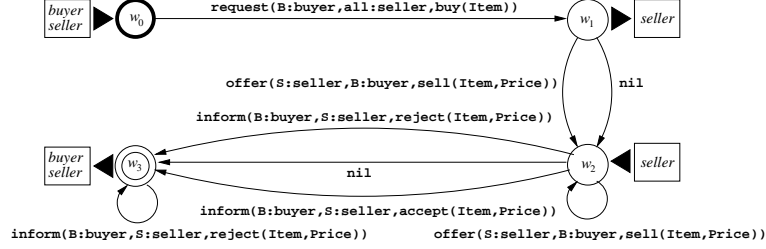


Fig. 1: Simple Agora Room Scene

scene for an agora room in which an agent willing to acquire goods interacts with a number of agents intending to sell such goods. This agora scene has been simplified – no auctions or negotiations are contemplated. The buyer announces the goods it wants to purchase, collects the offers from sellers (if any) and chooses the best (cheapest) of them. The simplicity of this scene is deliberate, in order to make the ensuing discussion and examples more accessible. A more friendly visual rendition of the formal definition is employed in the figure. Two roles, buyer and seller, are defined. The initial state w_0 is denoted by a thicker circle (top left state of scene); the only final state, w_3 , is represented by a pair of concentric circles (bottom left state). Access states are marked with a “►” pointing towards the state with a box containing the roles of the agents that are allowed to enter the scene at that point. Exit states are marked with a “►” pointing away from the state, with a box containing the roles of the agents that may leave the scene at that point. The edges are labelled with the messages to be sent/received at each stage of the scene. A special label “nil” has been used to denote edges that can be followed without any action/event.

We now provide a definition for e-institutions:

Definition 2. An e-institution is the tuple $\mathcal{E} = \langle SC, T, \mathbf{S}_0, \mathbf{S}_\Omega, E, \lambda_E \rangle$ where

- $SC = \{\mathbf{S}_1, \dots, \mathbf{S}_n\}$ is a finite, non-empty set of scenes;
- $T = \{t_1, \dots, t_m\}$ is a finite, non-empty set of transitions;
- $\mathbf{S}_0 \in SC$ is the root scene;
- $\mathbf{S}_\Omega \in SC$ is the output scene;
- $E = E^I \cup E^O$ is a set of arcs such that $E^I \subseteq WE^{\mathbf{S}} \times T$ is a set of edges from all exit states $WE^{\mathbf{S}}$ of every scene \mathbf{S} to some transition T , and $E^O \subseteq T \times WA^{\mathbf{S}}$ is a set of edges connecting all transitions to an access state $WA^{\mathbf{S}}$ of some scene \mathbf{S} ;
- $\lambda_E : E \mapsto p(x_1, \dots, x_k)$ maps each arc to a predicate representing the arc’s constraints.

¹ Roles in e-institutions are more than labels: they help designers abstract from individual agents thus defining a *pattern of behaviour* that any agent adopting that role ought to conform to. Moreover, all agents with the same role are guaranteed the same rights, duties and opportunities [4].

Transitions are special connections between scenes through which agents move, possibly changing roles and synchronising with other agents. We illustrate the definition above with an example comprising a complete virtual agoric market. This e-institution has more components than the above scene: before agents can take part in the agora they have to be admitted; after the agora room scene is finished, buyers and sellers must proceed to settle their debts. In Figure 2 we show a graphic rendition of an e-institution for our market. The scenes are shown

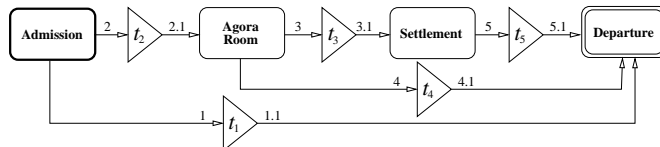


Fig. 2: E-Institution for Simple Agoric Market

in the boxes with rounded edges. The root scene is represented as a thicker box and the output scene as a double box. Transitions are represented as triangles. The arcs connect exit states of scenes to transitions, and transitions to access states. The labels of the arcs have been represented as numbers. The same e-institution is, of course, amenable to different visual renditions.

The predicates $p(x_1, \dots, x_k)$ labelling the arcs, shown above as numbers, typically represent constraints on roles that agents ought to have to move into a transition, how the role changes as the agent moves out of the transition, as well as the number of agents that are allowed to move through the transition and whether they should synchronise their moving through it. In the Agoric Market above, the arc label 3 is:

$$p_3(x, y) \leftarrow id(x) \wedge role(y) \wedge y \in \{seller, buyer\} \wedge \langle x, y \rangle \in Ags \quad (3)$$

that is, transition t_3 is restricted to those agents x whose role y is either *seller* or *buyer* – information on such agents is recorded in the set *Ags*. The complementary arc label 3.1 leaving transition t_3 is:

$$p_{3.1}(x, z) \leftarrow \langle x, y \rangle \in Ags \wedge y/z \in \{seller/receiver, buyer/payer\} \quad (3.1)$$

that is, those agents $\langle x, y \rangle \in Ags$ that moved into t_3 may move out of the transition provided they change their roles: *seller* agents in the **Agora Room** scene should become *receiver* agents in the **Settlement** scene, *buyer* agents should become *payer* agents.

2.1 Representing E-Institutions

We have represented our e-institutions in a logical formalism [6] implemented in Prolog [10], making them computer-processable. We show in Figure 3 our Prolog representation for the agora room scene graphically depicted in Figure 1 above. Each component of the formal definition has its corresponding representation. Since many scenes may coexist within one e-institution, the components are parameterised by a scene name (first parameter). The Θ and λ components of the definition are represented together in `theta/2`, where the second argument holds a list containing the directed edge as the first and third elements of the list and the label as the second element.

```

roles( agora, [buyer, seller] ).
states( agora, [w0, w1, w2, w3] ).
initial_state( agora, w0 ).
final_states( agora, [w3] ).
access_states( agora, buyer, [w0] ).
access_states( agora, seller, [w0, w2] ).
exit_states( agora, buyer, [w3] ).
exit_states( agora, seller, [w1, w3] ).
theta( agora, [w0, request( B:buyer, all: seller, buy(I) ), w1] ).
theta( agora, [w1, offer( S: seller, B: buyer, sell(I, P) ), w2] ).
theta( agora, [w1, nil, w2] ).
theta( agora, [w2, offer( S: seller, B: buyer, sell(I, P) ), w2] ).
theta( agora, [w2, inform( B: buyer, S: seller, accept(I, P) ), w3] ).
theta( agora, [w2, inform( B: buyer, S: seller, reject(I, P) ), w3] ).
theta( agora, [w2, nil, w3] ).
theta( agora, [w3, inform( B: buyer, S: seller, reject(I, P) ), w3] ).

```

Fig. 3: Representation of Agora Room Scene

Any scene can be conveniently and economically described in this fashion. E-institutions are collections of scenes in this format, plus the extra components of the tuple comprising its formal definition. In Figure 4 we present a Prolog representation for the agora market e-institution. Of particular importance are the arcs connecting scenes to transitions and vice-versa. In definition 2 arcs E are defined as the union of two sets $E = E^I \cup E^O$, E^I connecting (exit states of) scenes to transitions, and E^O connecting transitions to (access states of)

```

scenes( [admission, agora, settlement, departure] ).
transitions( [t1, t2, t3, t4, t5] ).
root_scene( admission ).          output_scene( departure ).
arc( [admission, w3], p1, t1 ).    arc( t1, p1.1, [departure, w0] ).
arc( [admission, w3], p2, t2 ).    arc( t2, p2.1, [agora, w0] ).
                                   arc( t2, p2.1, [agora, w2] ).
arc( [agora, w3], p3, t3 ).         arc( t3, p3.1, [settlement, w0] ).
arc( [agora, w1], p4, t4 ).         arc( t4, p4.1, [departure, w0] ).
arc( [agora, w3], p4, t4 ).
arc( [settlement, w3], p5, t5 ).    arc( t5, p5.1, [departure, w0] ).

```

Fig. 4: Representation of E-Institution

scenes. We represent the E^I arcs as `arc/3` facts: its first argument is a list which holds a scene and one of its exit states, the second argument holds the predicate (constraint) p_i which enables the arc, and the third argument is the destination transition. For simplicity, we choose to represent the arcs of E^O also as `arc/3` facts, but with different arguments: the first argument holds the transition, the second argument holds the constraint that enables the arc, and the third argument holds (as a list) a scene and one of its access states.

3 Norms in E-Institutions

We adopt a pragmatic notion of norm as the prescription of a set of actions that an agent is obliged to carry out during its participation in an e-institution enactment. In our definition below, the actions contemplated by our norms concern utterances that agents ought to issue, that is, messages that ought to be sent².

As identical utterances in different contexts (*e.g.*, saying “yes” to a waiter serving you more wine and saying “yes” to a police officer asking if you committed a crime) serve very different purposes and cause rather disparate obligations, our actions will be uniquely identified as the pair (\mathbf{S}, γ) where \mathbf{S} is a scene and $\gamma \in CL$ is an illocution from the agreed communication language [4]. The complete set of actions of an e-institution is given by the union of all utterances

² Other actions, such as manipulating data structures, updating internal beliefs, or moving the arm of a robot, can easily be accommodated if we associate a message (sent to an administrative agent) reporting that the action has been performed.

labelling the edges of each of its scenes [17]. Formally, given an e-institution $\mathcal{E} = \langle SC, T, \mathbf{S}_0, \mathbf{S}_\Omega, E, \lambda_E \rangle$, then $Actions^{\mathcal{E}}$, its set of actions, is defined as

$$\left\{ (\mathbf{S}, \gamma) \left| \begin{array}{l} \mathbf{S} \in SC, \mathbf{S} = \langle R, W, w_0, W_f, WA, WE, \Theta, \lambda \rangle, \\ (w, w') \in \Theta, \lambda((w, w')) = \gamma \end{array} \right. \right\}$$

That is, all labels $\lambda((w, w')) = \gamma$ on edges $(w, w') \in \Theta$ of each one of its scenes $\mathbf{S} \in SC$.

Our norms are defined as two finite sets of actions, one the set of preconditions, that is what causes the norm to be triggered, and the other the set of actions that agents are obliged to perform:

Definition 3. A norm is the pair $N^{\mathcal{E}} = \langle Pre, Obls \rangle$ where:

- $Pre \subseteq Actions^{\mathcal{E}}$ is the set of actions which must be performed (the preconditions) in order for the norm to be triggered.
- $Obls \subseteq Actions^{\mathcal{E}}$ is the set of actions that agents are obliged to perform after the norm has been triggered.

This definition is a simplification of that introduced in [12] – in particular we have dropped the boolean expression over variables. Another distinct feature of our formulation is the implicit logical operators in our norms: a norm $N^{\mathcal{E}} = \langle Pre, Obls \rangle$ where $Pre = \{a_1^{Pre}, \dots, a_n^{Pre}\}$ and $Obls = \{a_1^{Obls}, \dots, a_m^{Obls}\}$ is, implicitly, $(a_1^{Pre} \wedge \dots \wedge a_n^{Pre}) \rightarrow (a_1^{Obls} \wedge \dots \wedge a_m^{Obls})$.

Designers associate a possibly empty set of norms $\mathbf{N}^{\mathcal{E}} = \{N_0^{\mathcal{E}}, \dots, N_m^{\mathcal{E}}\}$ to their e-institutions. For the pair $\langle \mathcal{E}, \mathbf{N}^{\mathcal{E}} \rangle$ we introduce the term *normalised e-institution*. We show in Figure 5 below a sample norm for our e-institution of Figure 2. The norm prescribes the implications of an agent B playing the role

$$\left\langle \left\{ \left\{ (agora, inform(B : buyer, S : seller, accept(Item, Price))) \right\} \right\}, \left\{ (settlement, inform(B : payer, S : receiver, pay(Price))) \right\} \right\rangle$$

Fig. 5: A Sample Norm

of a buyer in the *agora* scene and sending a message to an agent S playing the role of a seller: the message informs that B accepts the offered *Price* for *Item*. If this holds then agent B is obliged to pay and should send a message in scene *settlement* informing S that *Price* will have been paid. We show our sample norm above represented in Prolog in Figure 6. We use the term `norm(Name, Pre, Obls)`

```
norm(n1, [(agora, inform(B:buyer, S:seller, accept(Item, Price))),
          (settlement, inform(B:payer, S:receiver, pay(Price)))]).
```

Fig. 6: Sample Norm in Prolog

to represent our norms in Prolog, where `Name` is a label to identify the norm, `Pre` and `Obls` are lists of pairs (`Scene, Illocution`) storing the actions of the preconditions and obligations, respectively.

3.1 Norm Verification of E-Institutions

An initial test designers need to perform is the well-formedness of a set of norms. This is straightforward: all we need to do is to check if the actions in the sets

Pre and *Obls* of every $N_i^{\mathcal{E}}$ appear as labels on the edges of a scene in \mathcal{E} . We also need to check if all scenes referred to indeed have been defined in \mathcal{E} .

A more useful check concerns the *feasibility* of a norm, that is, given an e-institution we want to know if the *Pre* actions of a norm will ever take place and if its *Obls* obligation actions will ever be fulfilled. We can verify this property by checking for paths within the scenes and transitions of an e-institution, thus trying to find at least one path connecting the initial state of the root scene to a final state of the output scene in which the actions of a norm appear as labels. The order of actions in norms is not important in our approach³: as long as the action takes place (*i.e.*, there is a label in a path) then we can tick the action off as being performable.

We show in Figure 7 a straightforward implementation of this approach.

```

1 feasible_actions(_, []).
2 feasible_actions(Path, Actions):-
3   Path = [(Scene, State) | _],
4   theta(Scene, [State, M, NewState]),
5   \+ member((Scene, NewState), Path),
6   member((Scene, M), Actions),
7   delete(Actions, (Scene, M), RestActions),
8   feasible_actions([(Scene, NewState) | Path], RestActions).
9 feasible_actions(Path, Actions):-
10  Path = [(Scene, State) | _],
11  theta(Scene, [State, M, NewState]),
12  \+ member((Scene, NewState), Path),
13  \+ member((Scene, M), Actions),
14  feasible_actions([(Scene, NewState) | Path], Actions).
15 feasible_actions(Path, Actions):-
16  Path = [(Scene, State) | _],
17  arc([Scene, State], _, Transition),
18  arc(Transition, [NewScene, NewState]),
19  feasible_actions([(NewScene, NewState) | Path], Actions).

```

Fig. 7: Program to Check Feasibility of Actions

Predicate `feasible_actions/2` builds a path in its first argument and gradually removes from the list of actions in its second argument those elements it finds labelling edges within the scenes of the e-institution. The path in the first argument is required to avoid loops. Line 1 shows the condition for successful termination: the list of actions is empty (and the contents of the path are irrelevant).

Clause 2 (lines 2–8) addresses the case when a Θ edge is to be followed but whose associated λ label is an illocution in one of the actions – in this case the matching action is removed (via built-in predicate `delete/3`) from the list of actions and the remaining actions are recursively examined. Clause 3 (lines 9–14) exploits a similar situation, but the illocution labelling the Θ edge does not occur in the list of actions – in this case, `feasible_actions/2` simply updates the path and carries on examining the list of actions. Finally, clause 4 (lines

³ We are aware of the fact that in some situations the order of actions is essential.

In [17] we put forth a more expressive definition of norms in which the order of events is taken into account. Our functionalities could be enhanced to account for the ordering of actions since the dialogues are followed in the order that they take place.

15–19) follows a transition from one scene to a new scene, carrying on the check for feasibility into the new scene.

Termination is guaranteed: either the program stops at line 1, when all actions are removed from the list `Actions` (2nd argument of `feasible_actions/2`) or the program terminates because it cannot find an alternative path (all paths are recorded in the 1st argument of `feasible_actions/2`) in which the actions in `Actions` may take place. Correctness is also guaranteed: if at least one action is not found in any of the dialogues of an institution, then the program fails – no new edges can be found and the list `Actions` is not empty, causing a failure. On the other hand, if the given list of actions is to be found in dialogues of the institution, clause 2–8 will remove each of them, one at a time.

The fragment of code above must be used twice for each norm: once to check the *Pre* actions and another time to check for the *Obls* actions. An initial value ought to be assigned to the path consisting of the root scene and its initial state. A top-level definition of the check for the feasibility of a norm is shown in Figure 8. Predicate `feasible/1` takes as its only parameter the name of a

```

1 feasible(Norm):-
2   norm(Norm,Pre,Obls),
3   root_scene(Scene),
4   initial_state(Scene,State),
5   feasible_actions([(Scene,State)],Pre),
6   feasible_actions([(Scene,State)],Obls).

```

Fig. 8: Predicate to Check Feasibility of Norms

norm and returns “yes” if that norm is feasible or “no” otherwise. It works by retrieving the definition of `Norm` (line 2), the root scene (line 3) and its initial state (line 4), then calling predicate `feasible_actions` for the action list `Pre` and `Obls`. Only if *both* `Pre` and `Obls` are feasible is that `Norm` is considered feasible.

Although the code above always terminates, its complexity is exponential in the worst case, as it tries all possible paths. This complexity can be reduced, however, via simple heuristics such as checking for all actions of each scene, using the scenes’ definitions to control the checking loop. For instance, if we check for all actions of a norm that should take place in a certain scene and we find that at least one of them is not found, then we can stop the verification as the norm is unfeasible.

We envisage two likely scenarios for norm verification. In the first scenario designers willing to create norms for an existing e-institution can verify if these new norms are feasible: designers may alter and change norms until they achieve feasibility. In the second scenario designers in possession of a norm which captures a desirable property of agents and their illocutions may “tinker” with an e-institution until it complies with the norm. The same feasibility verification can thus lead to changes in the norm, in the e-institution or in both, depending on the designers’ intention.

If we consider our actions to be ordered, then the code above has to reflect this. The execution control should be guided by the list of actions to be searched in the dialogues: for each action, check that it takes place in a dialogue, *in the order* they appear in the list `Actions`.

4 Norm Analysis of E-Institutions

Normalised e-institutions provide a hitherto unexplored approach to the analysis and engineering of multiagent systems: designers manipulate the normalised e-institution with a view to *extracting* sub-portions of it. These sub-portions are guaranteed to avoid or indeed cause specific obligations on those agents taking part in the original e-institution. The more limited e-institution(s) can be used as a guideline to synthesise agents which conform to the specification (as introduced in [6, 9]) but have restricted forms of behaviour.

Clearly, the removal of parts of an e-institution is a difficult and error-prone task and designers need support to perform it. We propose the use of *meta-programming* [13, 14] to help designers analyse and manipulate e-institutions with a view to extracting sub-portions of it in which certain properties hold. A meta-program is a program whose data denotes another (object) program, both of which are in the same language.

We have designed a meta-interpreter, shown in Figure 9, to build a list with those portions of the original e-institution used to compute a result. Predicate

```
1 meta((G,Gs),TmpEI,EI):-
2   meta(G,TmpEI,NewTmpEI),
3   meta(Gs,NewTmpEI,EI).
4 meta(G,EI,EI):-
5   system(G),
6   call(G).
7 meta(G,EITmp,EI):-
8   clause(G,Body),
9   update(EITmp,G,NewEITmp),
10  meta(Body,NewEITmp,EI).

11 update(EI,G,[G|EI]):-
12   collectable(G),
13   \+ member(G,EI).
14 update(EI,-,EI).

15 collectable(roles(-,-)).
16 collectable(states(-,-)).
. . .
```

Fig. 9: Program to Collect Portions of E-Institution

`meta/3` builds in its third argument a list with the components of the e-institution that were used in the proof of its first argument. The second argument is a temporary list with the components used so far in the proof and is initially assigned the empty list.

The first clause (lines 1–3) caters for a conjunction of goals (G, Gs) and recursively builds its list of goals used in the proof of G and uses it to build the list of goals of Gs . The second clause (lines 4–6) addresses the built-in predicates, those goals G that satisfy the built-in test `system/1`. The third clause (lines 7–10) handles user-defined predicates: a clause from the program is selected via the `clause/2` built-in (line 8) whose head matches G and its body is returned in `Body`. The goal G is then used to update (line 9) the list `EITmp` containing the portions of the e-institution used so far – predicate `update/3` defined in lines 1–14 inserts G as the head of its third argument if it is a collectable goal and

does not yet appear in the list. The body of the clause is recursively used with the updated result (line 10).

The collectable goals defined via the `collectable/1` predicate (lines 15 onwards) are all those used in the definition of an e-institution, such as `roles/2`, `states/2`, and so on. These are the goals required to completely define an e-institution and are the ones that should be collected during the execution of the meta-interpreter. If a goal is not collectable, then the second clause of `update/3` returns the same input e-institution.

4.1 Norm-Based Extraction

In order to extract sub-parts of the e-institution that make up a coherent whole, we ought to make sure an agent can join it and find its way from an initial state of the root scene to a final state of the output scene.

We have designed a program which captures the behaviours of a generic agent within an e-institution. This program is shown in Figure 10: predicate `loop/1` (lines 1–4) gathers information and makes an initial call to its auxiliary `loop/2` (lines 5–19) predicate. Predicate `loop/1` has only one argument `Ag`, an agent

```

1 loop(Ag):-
2   root_scene(Scene), initial_state(Scene,State),
3   role_scenes(Scene,Roles), member(Role,Roles),
4   loop([Scene,State,Role,nil],Ag).
5 loop([(Scene,State,_,_)|_],_):-
6   output_scene(Scene),
7   final_states(Scene,States),
8   member(State,States).
9 loop(Path,Ag):-
10  Path = [(Scene,State,Role,_)|_],
11  theta(Scene,[State,M,NewState]),
12  illocution(Role,Ag,M,AcM),
13  \+ member((Scene,State,Role,AcM),Path),
14  loop([(Scene,NewState,Role,AcM)|Path],Ag).
15 loop(Path,Ag):-
16  Path = [(Scene,State,Role,_)|_],
17  arc([Scene,State],_,Tr), arc(Tr,_,[NewScene,NewState]),
18  roles(NewScene,Roles), member(Role,Roles),
19  loop([(NewScene,NewState,Role,nil)|Path],Ag).

20 illocution(Role,Ag,M,M):-
21   M =.. [_ ,Ag:Role,_,_] ; M =.. [_ ,_,Ag:Role,_,_].
22 illocution(_,_,_,nil).

```

Fig. 10: Generic E-Institution Agent

identifier. It obtains the initial state in the root scene (line 2), then selects a role (line 3) from the possible roles of the root scene. It then makes an initial call to its auxiliary predicate `loop/2` which defines a loop.

The first argument of predicate `loop/2` is a list of tuples (`Scene,State,Role,Illocution`) storing a path an agent can follow within the e-institution and the second argument is the unique identification of the agent. The first clause (lines 5–8) captures the termination condition when a final state of the output scene is reached. The second clause (lines 9–14) addresses Θ edges within a scene, making sure that the new state and message are not part of the current path built. Finally, the third clause (lines 15–19) caters for transitions between two

scenes: the transitions out of the current scene and into the new scene are followed in line 13, a role is picked for the new scene (line 14) and the loop carries on recursively.

The second clause of predicate `loop/2` makes use of an auxiliary predicate `illocution/4` (lines 20–22). This predicate obtains in its fourth argument the actual message sent or received by an agent incorporating role `Role`: it may send the message (first case of line 21), receive the message (second case of line 21) or none of them (line 22 – a “`nil`” illocution is returned), depending on whether its role matches the one specified in the λ label.

We can put our meta-interpreter above to use in order obtain the parts of an e-institution that guarantee that a norm will hold, by using the query

```
?- meta((loop(ag1),feasible(n1)), [],EI).
```

asking for the portions `EI` of the e-institution in which both `loop(ag1)` and `feasible(n1)` hold, that is, the subparts of the e-institution required for an agent to find its way into and out of it and such that norm `n1` (defined in Fig. 6) holds.

If we use the query above with the definitions of Figures 3 and 4, then we obtain in `EI` the parts of the e-institution definition required to prove that norm `n1` is feasible, that is, the portions of the e-institution required to allow an agent to correctly navigate its way into and out of it and, in addition to that, the parts ensure that the norm has both its preconditions and obligations fulfilled. We show in Figures 11 and 12 the visual rendition of the fragments of,

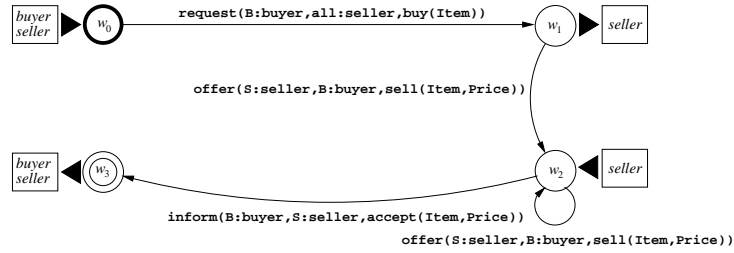


Fig. 11: Portion of Agora Room Scene

respectively, the agora scene and the agoric market e-institution obtained with the query above. The scene fragment shows the edges and labels that should

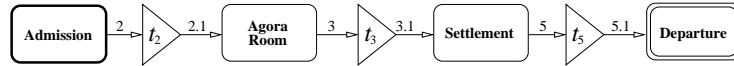


Fig. 12: Portion of Agoric Market E-Institution

be followed by agents in order for the pre-conditions of the norm to hold. The fragment of the e-institution shows those scenes that ought to take place in order for the obligations to be fulfilled – the alternative paths that bypass the agora room are eliminated.

Alternatively, we can obtain the portions of an e-institution that allow agents to join in and leave, but *avoiding* the conditions that would bind them to a norm. In order to do that, we ought to get hold of a proper portion of the e-institution (*i.e.* one that allows an agent join in and leave it) and in which the pre-conditions

of the norm does not hold. The auxiliary definition of Figure 13 captures the conditions when a norm cannot be triggered. The definition is similar to that

```

1 untriggered(Norm):-
2   norm(Norm,Pre,Obls),
3   root_scene(Scene),
4   initial_state(Scene,State),
5   \+ feasible_actions([(Scene,State)],Pre).

```

Fig. 13: Test for Untriggered Norms

in Figure 8, but here the `feasible_actions/2` predicate is used in its negated form. Moreover, only the preconditions of the norm are tested: an untriggered norm is one whose preconditions do not occur in the e-institution.

The query below obtains the portions of the e-institution that allow an agent to join in and leave it, but avoids triggering the norm by causing its preconditions:

```
?- meta((loop(ag1),untriggered(n1)), [],EI).
```

that is, it obtains in EI the parts of the e-institution used to allow agent `ag1` to navigate it but these parts do not trigger the preconditions of norm `n1`. If we use the query above with the e-institution of Figures 3 and 4, then we get the fragments shown in Figures 14 and 15 (represented in their visual form).

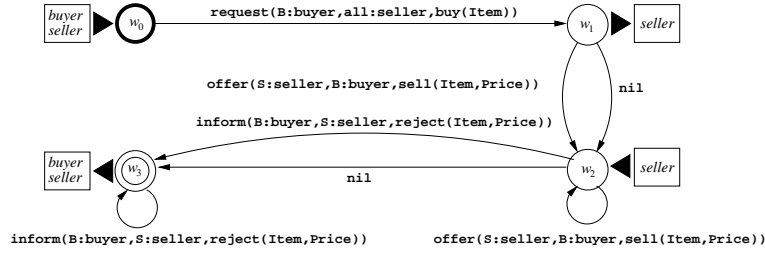


Fig. 14: Another Portion of the Agora Room Scene

Figure 14 shows the agora scene but the edge labelled with the message that

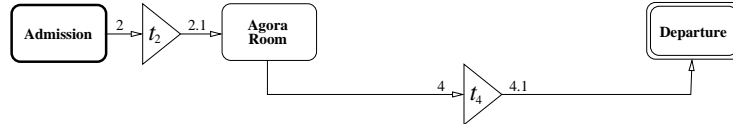


Fig. 15: Another Portion of the E-Institution

would trigger norm `n1` has been removed. This fragment of the agora scene becomes part of the e-institution depicted in Figure 12. We have obtained only those parts used to go from the root scene to the output scene via one of the many existing paths.

Our formalisation of e-institutions exploits non-determinism to represent the many different behaviours agents are allowed to have. When an e-institution is analysed using our queries above, only *one* path in and out of the e-institution is actually pursued. We can, however, exhaustively examine all paths obtaining all sub-parts of the e-institution in which a norm is fulfilled or avoided. Our approach allows any combination of any number of norms to be fulfilled and/or avoided.

4.2 Norm-Aware Synthesis of Agents

In [6, 9] we have shown how we can synthesise simple agents conforming to a given specified e-institution. We have also shown how these simple agents can be further customised into more sophisticated software. We notice that the restricted e-institutions obtained via our approach explained above can be used to synthesise agents – these agents will correctly follow the e-institution but will pursue paths in which norms can be triggered (and fulfilled) or paths in which norms cannot be triggered.

We envisage a scenario in which an initial normalised e-institution is manipulated using the approach described above, giving rise to a number of alternative e-institutions. Each of these alternative e-institutions is fully compatible with the original one but they offer particular “views” in which norms are fulfilled or avoided. The alternative e-institutions can be used to synthesise agents that will adopt norm-avoiding or norm-fulfilling behaviour.

This approach is depicted in the diagram of Figure 16 below: an initial e-institution \mathcal{E} is used to extract (simple arrow) a repertoire of e-institutions \mathcal{E}'_i each of which has particular features of avoiding or fulfilling norms. Each of these extracted e-institutions is used to synthesise agents $\Pi_{[i]}$ (double arrows). The synthesised initial agents are then customised differently as $\Pi_{[i,j]}$ (triple

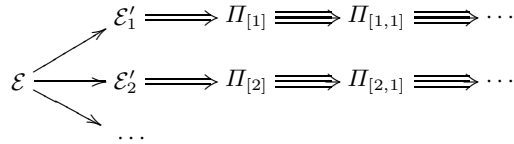


Fig. 16: Extraction, Synthesis & Customisation (triple arrows). The customised agents can take part in an enactment of e-institution \mathcal{E} as they will be in full compliance with it, but the agent will be adverse to particular norms or eager to fulfil them.

5 Conclusions, Related Work and Directions of Research

We have presented a formal definition of norms and shown how norms can be incorporated in electronic institutions and employed to verify properties both of norms and electronic institutions. We have also introduced automatic means to obtain portions of an e-institution in which norms are guaranteed to hold and portions in which norms can be safely avoided.

Clearly, not all kinds of norms can be represented in our approach. In particular, we focus on utterances: the only events we consider are those of issuing messages. Additional events associated to, for instance, sensors or data structures, although important in many applications of multiagent systems, are not considered in our approach.

The scenario we contemplate is one in which an electronic institution is endowed with a layer of administrative (or institutional) agents, the *governor agents*. These agents work as proxies of heterogeneous (external) agents that will join in in the enactment of the institution. The governor agents guarantee that the external agents will follow the specifications of the institution, sending the

appropriate messages in the prescribed order. The governor agents, plus a team of other administrative agents, form a *social layer* to the institution [15]. Issues of trust and sincerity are confined to the communication between the governor agent and its external agent. Various mechanisms can be put in place to prevent these issues from spreading to other parts of the institution.

Electronic institutions provide an ideal scenario within which alternative definitions and formalisations of norms can be proposed and studied. In [12] we find an early account of norms relating illocutions of an e-institution. In [16] we find a first-order logic formulation of norms for e-institutions: an institution conforms to a set of norms if it is a logical model for them.

Our work is an adaptation and extension of [12] but our approach differs in that we do not explicitly employ any deontic notions of obligations [1]. Our norms are of the form $Pre \rightarrow Obls$, that is, if Pre holds then $Obls$ ought to hold. The components of Pre and $Obls$ are utterances, that is, messages the agents participating in the e-institution send. This more pragmatic definition fits in naturally with the view of e-institutions as a specification of virtual environments which can be checked for properties and then used for synthesising agents [6, 9].

We represent e-institutions in a non-deterministic fashion: all possible behaviours of agents that will perform within it are captured. However, this feature causes an exponential number of possibilities to be considered when verifying and analysing e-institutions – the behaviours of the agents are paths of a non-deterministic finite-state machine. The functionalities described in this paper all have the same undesirable property: in the worst case, their computational complexity is exponential as they have to consider *all* possible behaviours.

Rather than extracting a complete e-institution as explained in Section 4.1, we can offer a similar functionality that collects just a single path (or a set of paths) that agents may follow in order to fulfil a norm or avoid it. Such a path can be supplied (in various alternative formats) to heterogeneous agents wanting to join the e-institution or to institutional agents looking over the enactment of an e-institution. The paths provide an agenda to help agents deliberate when given choices of behaviour.

We would like to include *prohibitions* in our norms as a set of actions that ought not to take place in an e-institution. Prohibitions would allow norms and e-institutions to be checked for *consistency*: an agent cannot be obliged to perform an action and simultaneously be prohibited from doing it. Furthermore, we have explored in [17] a more expressive notion of norms in which the ordering of the events is taken into account and there can be arbitrary constraints on the variables of our illocutions. Ideally this richer formalisation should be accompanied by algorithms and tools to verify properties and perform distinct analyses in electronic institutions. We are currently working on means to automate the verification and analysis of these more expressive norms.

Acknowledgements: Thanks are due to J. Rodríguez-Aguillar and M. Esteva for their comments and suggestions, and to Seumas Simpson for proofreading earlier versions of this document. Thanks are also due to the anonymous reviewers whose comments helped improving this paper. Any remaining mistakes are the author's responsibility only.

References

1. Dignum, F.: Autonomous Agents with Norms. *Artificial Intelligence and Law* **7** (1999) 69–79
2. López y López, F., Luck, M., d’Inverno, M.: Constraining Autonomy Through Norms. In: *Proceedings of the 1st Int’l Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, ACM Press (2002)
3. Verhagen, H.: Norm Autonomous Agents. PhD thesis, Stockholm University (2000)
4. Esteva, M., Rodríguez-Aguilar, J.A., Sierra, C., Garcia, P., Arcos, J.L.: On the Formal Specification of Electronic Institutions. Volume 1991. Springer-Verlag (2001)
5. Rodríguez-Aguilar, J.A.: On the Design and Construction of Agent-mediated Electronic Institutions. PhD thesis, Institut d’Investigació en Intel·ligència Artificial (IIIA), Consejo Superior de Investigaciones Científicas (CSIC), Spain (2001)
6. Vasconcelos, W.W., Robertson, D., Sierra, C., Esteva, M., Sabater, J., Wooldridge, M.: Rapid Prototyping of Large Multi-Agent Systems through Logic Programming. *Annals of Mathematics and A.I.* **41** (2004) Special Issue on Logic-Based Agent Implementation.
7. Fuchs, N.E.: Specifications are (Preferably) Executable. *Software Engineering Journal* (1992) 323–334
8. Lloyd, J.W.: Practical Advantages of Declarative Programming. In: *Joint Conference on Declarative Programming, GULP-PRODE’94*. (1994) Invited Paper.
9. Vasconcelos, W.W., Sierra, C., Esteva, M.: An Approach to Rapid Prototyping of Large Multi-Agent Systems. In: *Proc. 17th IEEE Int’l Conf. on Automated Software Engineering (ASE 2002)*, Edinburgh, UK, IEEE Computer Society, U.S.A (2002)
10. Apt, K.R.: *From Logic Programming to Prolog*. Prentice-Hall, U.K. (1997)
11. Hopcroft, J.E., Ullman, J.D.: *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, U.S.A (1979)
12. Esteva, M., Padget, J., Sierra, C.: Formalizing a Language for Institutions and Norms. Volume 2333. Springer-Verlag (2001)
13. Hill, P.M., Gallagher, J.: Meta-Programming in Logic Programming. In: *Handbook of Logic in Artificial Intelligence and Logic Programming*. Volume 5., Oxford University Press (1998) 421–498
14. Sterling, L., Shapiro, E.: *The Art of Prolog: Advanced Programming Techniques*. 2nd edn. MIT Press (1994)
15. Esteva, M., Rosell, B., Rodríguez-Aguilar, J.A., Arcos, J.L.: AMELI: an Agent-Based Middleware for Electronic Institutions. In: *Proc. 3rd Int’l Joint Conf. on Autonomous Agents & Multi-Agent Systems (AAMAS)*, New York, U.S.A., ACM Press (2004)
16. Ibrahim, I.K., Kotsis, G., Schwinger, W.: Mapping Abstractions of Norms in Electronic Institutions. In: *12th. Int’l Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE’03)*, Linz, Austria, IEEE Computer Society (2003)
17. Esteva, M., Vasconcelos, W., Sierra, C., Rodríguez-Aguilar, J.A.: Verifying Norm Consistency in Electronic Institutions. In: *Proc. AAAI-04 Workshop on Agent Organizations: Theory and Practice*, San Jose, California, U.S.A., AAAI Press (2004)